

Tasking Design Descriptions

A guiding rule of software development is that no one likes to document their code. Here's how to make the paperwork less painful.

Mention documentation to engineers, and their eyes become glassy, their pupils dilate, and their palms sweat. Documentation is the ultimate bugaboo for all of us. But, as the mildly scatological cartoon in many offices reminds us, "The job's not done until the paperwork is done." Writing is easy, if you know where to start and what you want to say. This article will offer some guidelines to make writing tasking design documents easier and much less painful.

Several years ago, we encountered an unusual problem. One of our customers insisted that we change from one real-time executive to another. Our system had been in use for some time and worked well. To make the change, we had to map one set of tasking abstractions to another. No problem, we thought, we have good documentation. Our documentation, however, had almost nothing to say about the tasks,

interrupt-service routines (ISRs), and their interactions. The only place we could find this information was in the source listings. Needless to say, the process was slow and labor intensive. The service calls were easy to find; finding the single bit embedded in a table took a lot longer.

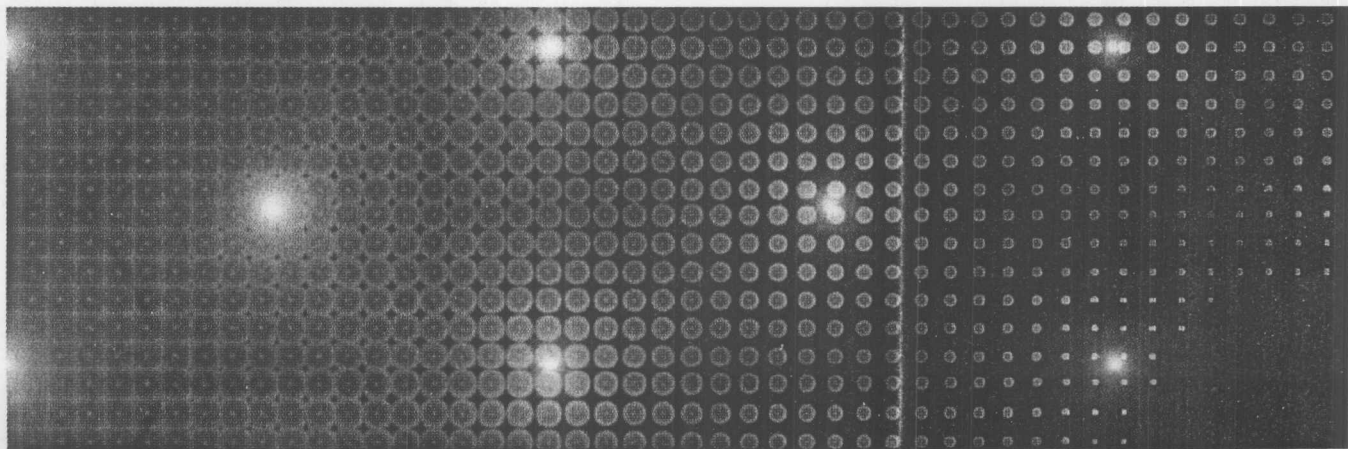
Shortly after the conversion job, the boss asked me to look at a large design document (about five inches thick) that supposedly described a system of some 120 tasks and several dozen ISRs. It contained very little useful information. Based on our conversion experience, we reduced that monster design document to a fraction of its original size, with considerably more useful information. Since then, we've used this method to describe a number of tasking designs, some as small as five tasks, but most somewhat larger. Hardware system engineers, customers, performance analysts, testers, and maintenance engineers have all commented favorably on this method.

MOVING AND SHAKING

We begin by drawing a graphical description of the tasking design using the symbols in Figure 1. Boxes or squares represent physical devices. Interrupts to ISRs are shown by broken lines, and ISRs shown as circles. Also, we'll use circles for tasks, processes, and other asynchronously executed software. Queues of commands or buffers will be shown with a fat letter I and semaphores with a skinny letter I. Optional time-outs will be represented with a small triangle, and data stores or tables will be shown by a 'container' silhouette. A shared resource, which needs concurrent access protection, is shown by a rectangle.

There is nothing sacrosanct about these symbols. You may use whatever is appropriate on whatever CASE tool you choose. You can add symbols, if you like, to represent event flags, pipes, mailboxes, and so on.

Each entity should be labeled or



named. I suggest using long and short form names as used in the program source code for each entity. Data-flow lines should be drawn to or from devices and buffers, and from tasks to buffers, or tasks to tasks as appropriate. Interrupts are drawn from devices to their associated ISRs. Signals from one task to another can be shown as interrupts, since that is what the signal looks like. Each line should be labeled or named.

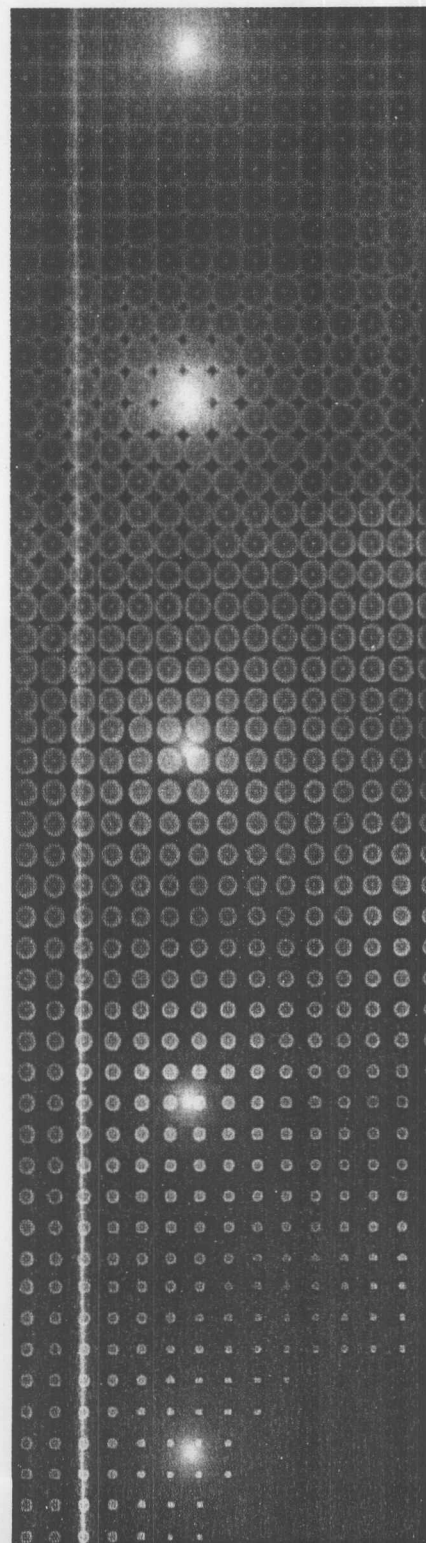
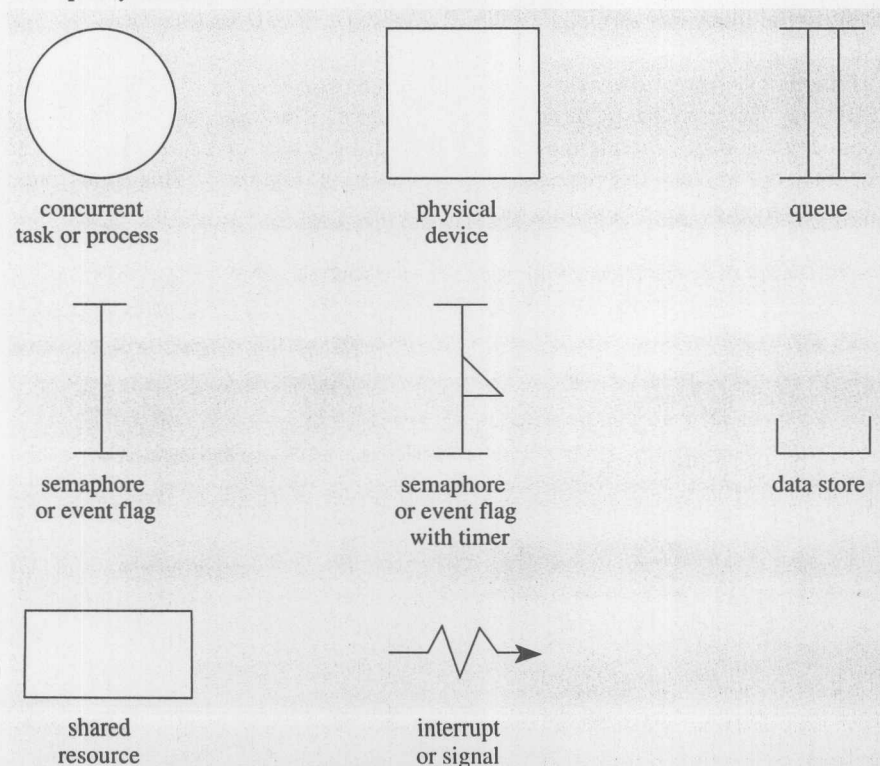
Explicit controls should also be drawn. For example, task A activates task B and later suspends and resumes it. A labeled dotted or dashed line from task A to task B is adequate, even though these controls are really service calls to the executive. Hidden tasks, some times called demons, should be shown. Several well-known real-time

executives use their own tasking services to implement other tasking features. For example, one executive for the 68HC11 uses an executive task to field timer interrupts that in turn activates application-level periodic tasks. These hidden tasks consume overhead and can cause interrupt latency problems. By explicitly documenting these tasks, maintenance engineers and users are warned about nonapplication tasks that could cause problems.

DOCUMENTING THE PROCESS

Once the diagram is complete, or nearly so, we can begin to document what's going on. For every line, entity, or element on the diagram, you should have one or two sentences describing that element. More may be written as you see fit. In

FIGURE 1
Example symbols.



Tasking Design Descriptions

Some ISRs may be invoked by tasks, and some devices may generate more than one interrupt for each I/O request.

In addition, three of the element types, tasks, ISRs, and shared resources, have key features you'll want to document. Let's start with tasks. For each task (or task type if a family of tasks is used), describe the following:

- Name—give both the long English name and source code name.
- Purpose—state what this task does. Leave the details to the detail design description or source code.
- Frequency—state the frequency or period of execution. For periodic tasks, hard-deadline tasks, or asynchronous tasks with response time requirements, the period needs to be identified. Minimum interarrival time of aperiodic tasks should be stated. Operational profiles may be useful here.

- Execution time—depending on your needs, give some estimate of or limitation on CPU use or time.
- Sequencing or control—state all conditions under which this task starts, stops, or blocks. Identify the agents responsible. If the task never terminates, say so. Identify all blocking points in the task. These will give insight into rescheduling points.
- Priority—state the priority of the task. If the priority is variable, state the values or ranges and give the mechanism and reason.
- Input and output—identify top-level data flowing in and out of the task.
- Blocking time—if the task uses a shared resource, give the amount of time the resource is locked by the task.
- Other executive services—executive or kernel services not otherwise given in the previous items should be called out.

ISRs receive the same treatment as tasks:

- Name—do the same as for tasks.
- Purpose—state what this ISR does. Identify the interrupts serviced by this ISR.
- Frequency—remember that some ISRs may be invoked by tasks, and some devices may generate more than one interrupt for each I/O request.
- Execution time—give execution times for each kind of device and interrupt.
- Sequencing or control—state all con-

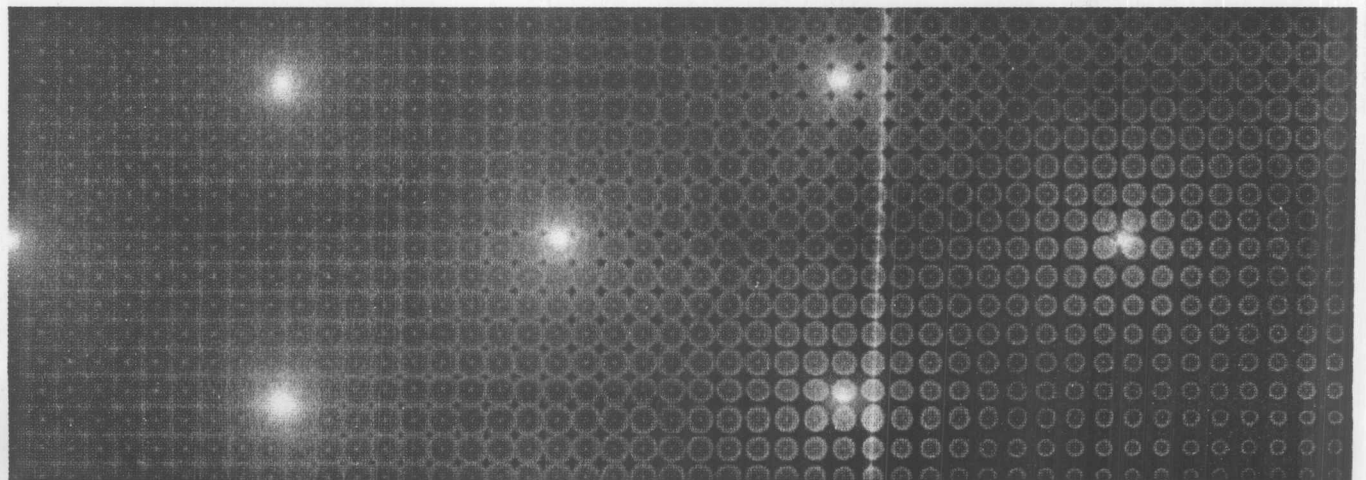
ditions, including hardware and software interrupts, that start this ISR. State the exit destination of this ISR. If this ISR changes the interrupt lock-out, enables, or priorities, include that description. Describe the execution environment and any restrictions on reentrancy.

- Priority—state the assigned priority level if supported by the hardware.
- Input and output—identify the inputs and outputs; usually these are commands and status data and not application data.
- Executive services—be sure to describe specific kernel services used by the ISR since they may not be explicitly shown on the diagram.

The last elements needing special attention are shared resources:

- Name—give the name of the resource.
- Purpose—state the purpose of the resource.
- Synchronization mechanism—identify the method used to serialize access to this resource; for example, semaphore, critical section, and so on. Identify time-out intervals if used.

Let's illustrate this method by describing part of a simple system as shown in Figure 2. This sub-system consists of an I/O device server task and its associated device interrupt-service routine.



- Name—`device_task` (`devtsk`).
- Purpose—the `device_task` manages I/O requests from application tasks by **PENDING** on the `request_queue`. When the `device_task` receives a request, it **WAITS** on the `request_sema` semaphore until the `i/o_req` register is free. When the `i/o_req` register is free, the task writes the request data in the `i/o_req` register and **SIGNALS** the `device_isr`. The task does a timed **WAIT** on the `i/o_complete` semaphore. When the `device_isr` **SIGNALS** the `i/o_complete` semaphore or the time-out occurs, the kernel restarts the `device_task`. The `device_task` reads the `status_ind` register and restarts the associated application task. The `device_task` **PENDS** on the `request_queue` for the next I/O operation.
- Frequency—aperiodic. `Device_task` is driven at the frequency of the requesting tasks.

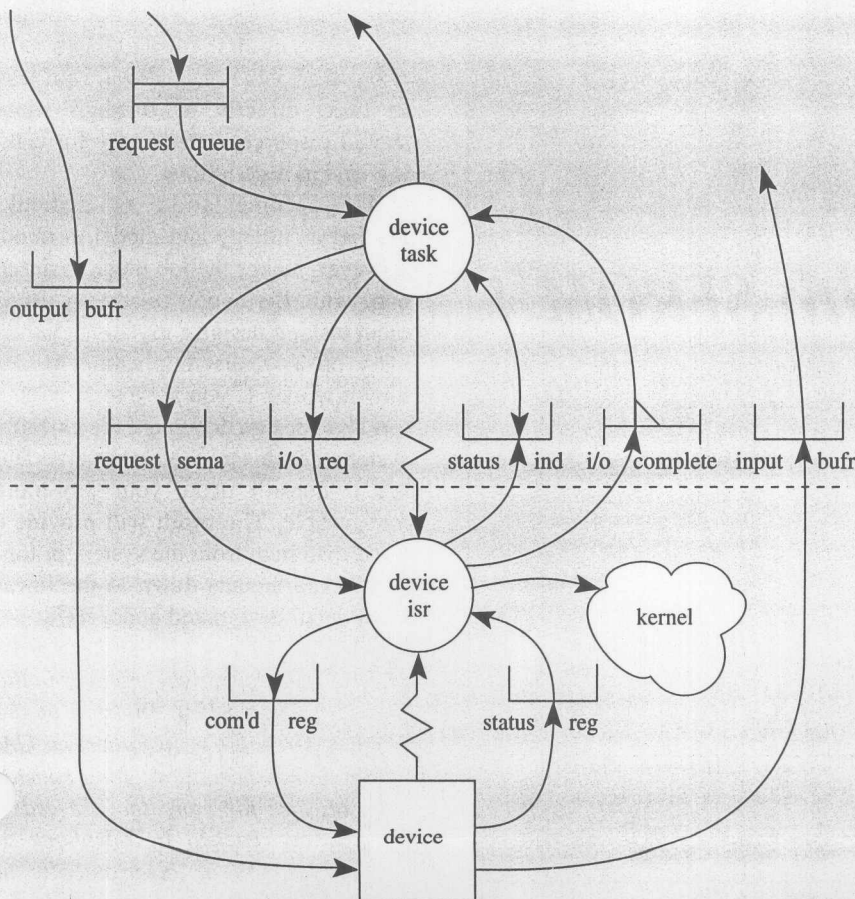
- Execution time—the average execution time of this task is 830 microseconds including kernel service times.
- Sequencing—this task is started by the initialization task, and it never terminates. This task may be blocked on the `request_queue` and the `request_sema`, and it will block on the `i/o_complete` semaphore. The signal operation on the `device_isr` will cause the `device_task` to be interrupted, and it may be delayed in its execution by the kernel dispatching another task. Restarting the requesting task may block the `device_task`.
- Priority—this task operates at priority level 3. A higher priority may result in excessive I/O operations and delays to application tasks, while a lower priority level may result in excessive queuing delays.
- Input and output—this task receives

The ISR manages the device I/O by setting up command registers and interpreting device status.

I/O requests from the request queue and status data from the `device_isr` and sends I/O commands to the `device_isr`.

- Blocking time—N/A.
- Other executive services—N/A.
- Name—`device_isr`.
- Purpose—this ISR manages the device I/O by setting up command registers and interpreting device status.
- Frequency—this ISR is driven by the frequency of requests from the associated device task.
- Execution time—this ISR takes a total of 67 microseconds to service one request and its interrupt.
- Sequencing and control—this ISR is started by a signal interrupt from the `device_task` and an interrupt from the device. It exits to the kernel interrupt return entry point. It **SIGNALS** the `request_sema` after reading the `i/o_req` register and **SIGNALS** the `i/o_complete` semaphore when the I/O operation is complete.
- Priority—`device_isr` operates at interrupt priority level 2, channel priority level 5.
- Input and output—`device_isr` receives I/O request data from the `device_task` and status data from the device. It sends commands to the device and status indications to the `device_task`.
- Executive services—N/A.

FIGURE 2
detailed example.



To reduce the clutter in Figure 2, we have not shown labels on the lines,

Tasking Design Descriptions

but you should label all the lines on your drawings.

These guidelines are powerful and useful. You write only what you need to write, no more and no less. Engineers often don't know what to write and once started don't know when to quit. As a result, real information content is low, and the resulting document becomes bulky, unreadable,

and unread.

When engineers first attempt to use this method of tasking design description, their diagrams tend to be too simple. That is, they don't have enough elements (graphics) to capture the essence of the design. The details on physical interfaces are usually ignored or forgotten. These details are important as they tend to drive all other inter-

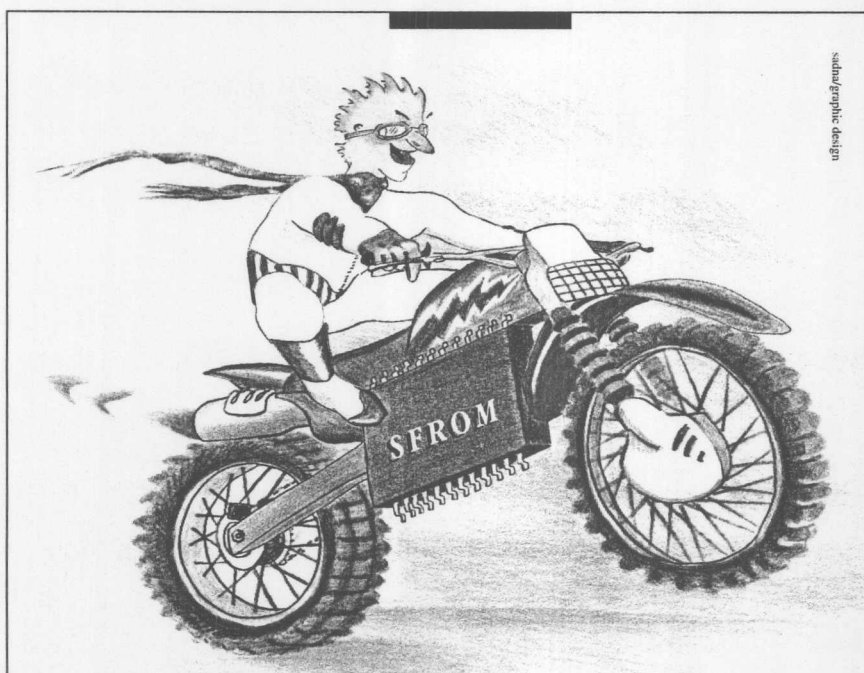
nal task sequencing. Faults, errors, and exception conditions are usually ignored but often constitute an important element of the design.

For example, on a recent application, engineers described an existing system as having only two interrupts. A closer examination showed 23 interrupts, many of which had a direct bearing on the application. Only a few interrupts dealt with internal error conditions that halted the machine. Even this detail is needed, however, for complete understanding of the system. When the additional interrupts and the associated ISRs were added, a much clearer picture of the system operation emerged.

The written descriptions, especially those parts involving the sequencing and control, should provide hints about what to add to make the drawings complete. It is possible to have too much detail, of course. For example, data-flow lines aren't needed for every data unit. Other kinds of design documentation may supply that detail. As a rule, only the fact that data flows from task to task, directly or through some shared resource, is necessary for tasking design descriptions.

All documentation is difficult. Having a strategy and model in mind, however, makes the job much easier to cope with. If you don't normally document tasking designs, give this method a try on a smaller application to see how it works. If you have to provide a top-level design document for a sizable system, use this method as a starter and see if doesn't make your job more manageable. The result will provide a good road map from the system or top-level requirements down to the lower level detail design and code. **ESP**

M.F. Moon is principal engineer in the Software Engineering Division at the Hughes Aircraft Co. in Fullerton, CA, where he has been developing embedded software for more than 27 years. He can be reached by telephone at (714) 732-3569 or electronically at m.moon@ieee.org.



Easy riding *with* SFROM™!

The fastest way from design to production

Tired of EPROM's erasing nightmare?

Challenged by Flash's in-circuit updatability?

Hampered by EEPROM's limited capacity?

Smart Flash ROM (SFROM) is the optimal re-programmable device for non-volatile memory that enables quick and easy continuous write and rewrite without removal and without UV erase.

SFROM is the only re-programmable device for non-volatile memory that incorporates large flash storage capacity *plus* in circuit remote re-programming capability with absolutely no design efforts.

SFROM incorporates:

- 256 Kbit to 16 Mbit flash memory
- Self-loading peripheral memory for firmware and/or data storage
- In circuit self re-programmability
- On-chip UART for serial interface
- I²C standard Interface

Call or fax:

EUROM
FlashWare
Solutions

See us in booth
208

- **Headquarters:** EUROM FlashWare Solutions Ltd.
Atidim Industrial Park Bldg. 1, P.O.B 58032, Tel Aviv 61580, Israel
Tel: +972-3-490 920 Fax: +972-3-490 922
- **USA Office:** EUROM FlashWare Solutions Inc.
4655 Old Ironsides Drive, Suite 200, Santa Clara, CA 95054
Tel: 408-748-9995 Fax: 408-748-8408

CIRCLE # 22 ON READER SERVICE CARD